Quantum algorithms for secure energy grids

Niels Neumann (TNO)

Agenda

Energy grid challenges

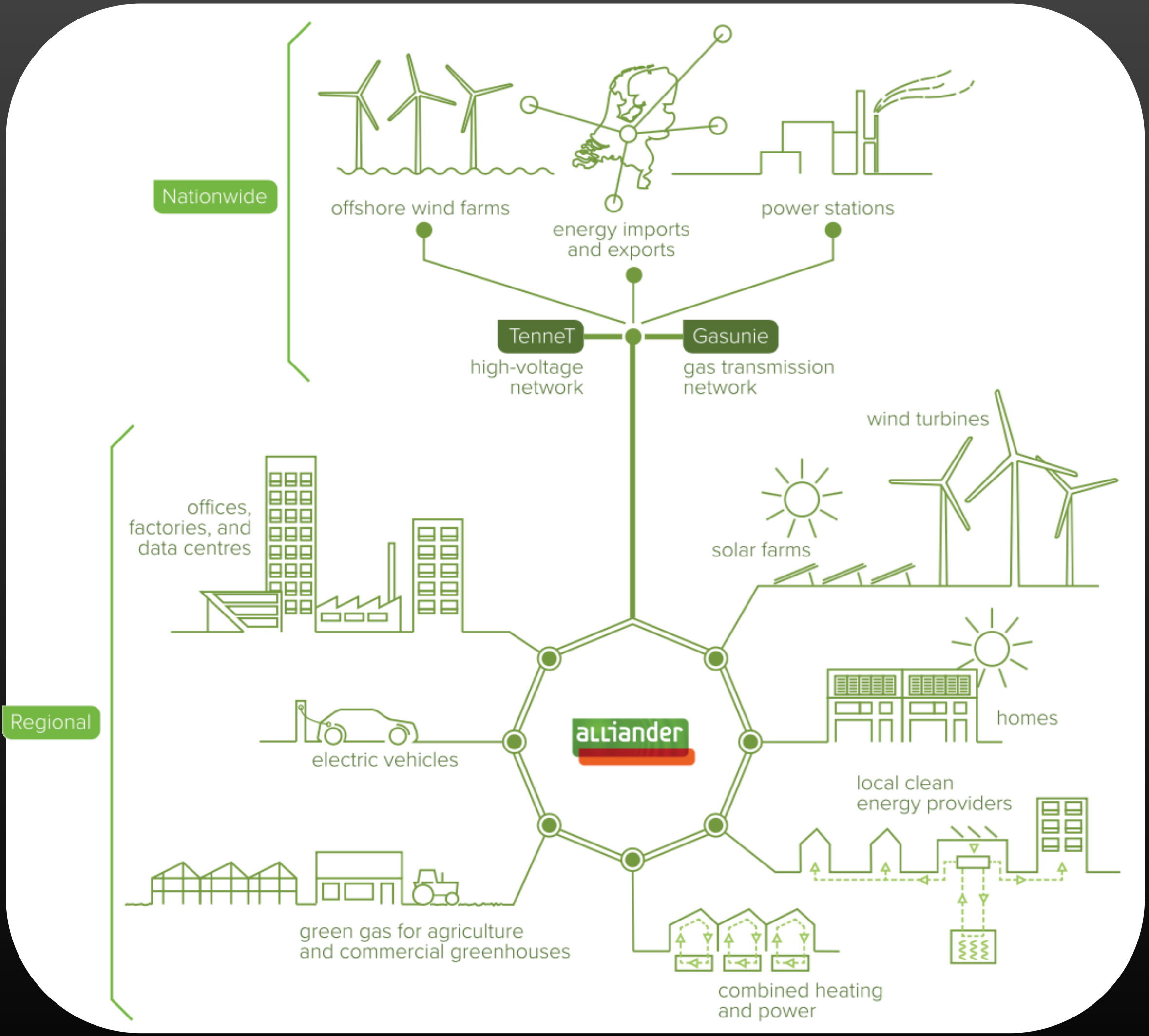
Gate-Based Quantum Solution

Quantum Annealing Solution

The energy grid of Alliander

Tasked to install at least as many assets in the next 10 years as in the last 100 years

Shortage of technicians and supplies



Customer service

Ready for further expansion

Prevent outages

Troubleshoot fast

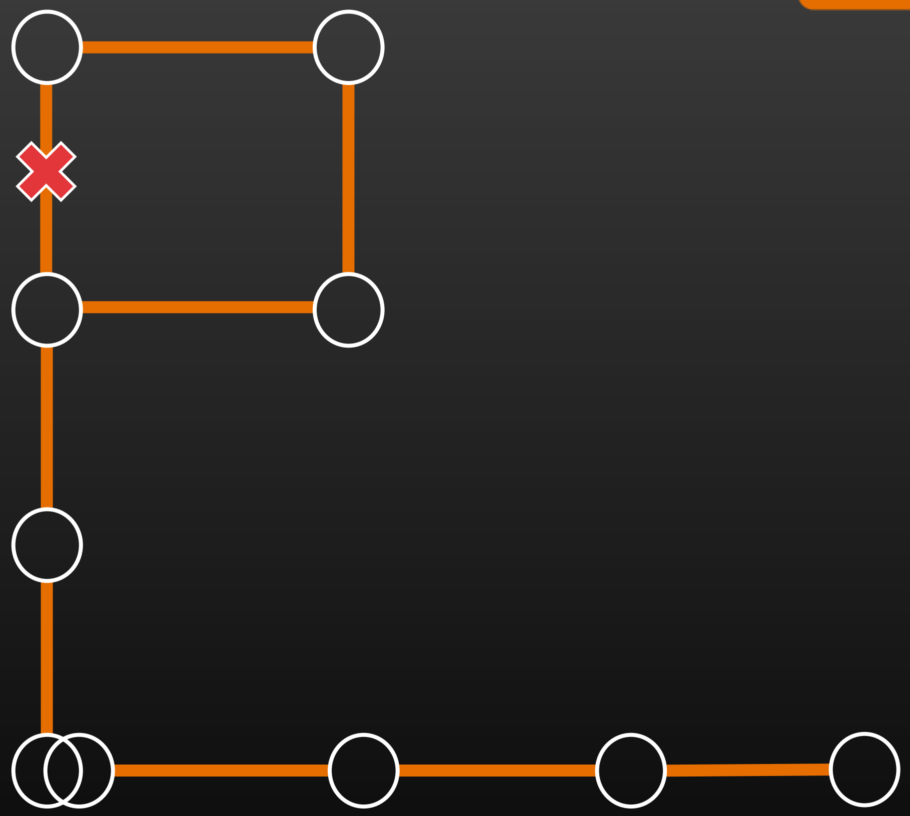
N-1 Challenge

The N-1 principle

If one asset fails, then it must be possible to resolve the failure by using the remaining assets in the network.

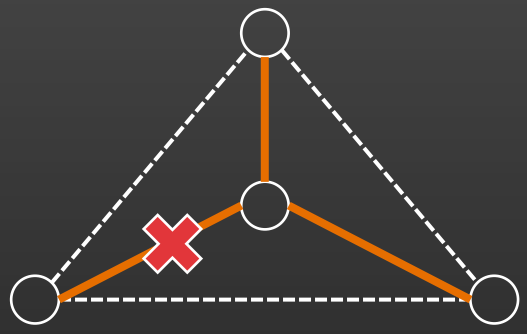
N-1 Challenge

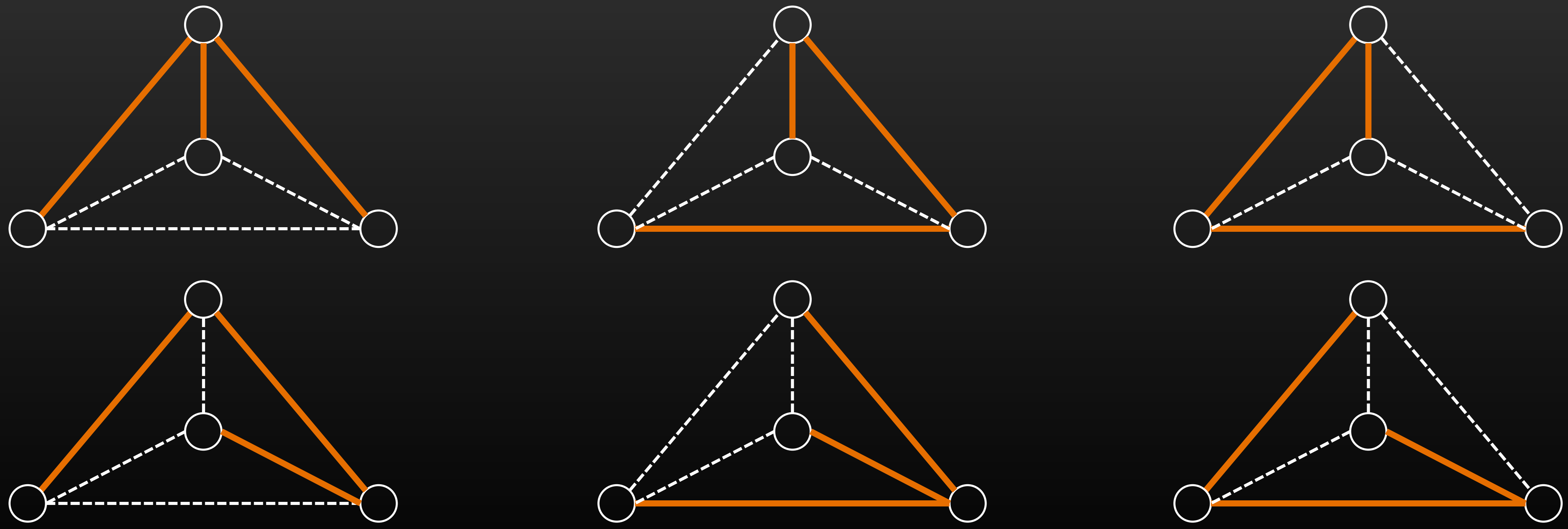
- Given a power network with edges (cables) labelled as active or inactive.
- Upon active edge failure, find a reconfiguration such that
  - The network is re-connected, with no cycles.
  - At most k switches are applied.
  - Load-flow constraints are met.
- We say that a network is "N-1 compliant" if a reconfiguration exists for all active edges.



N-1 Challenge

Example: reconfigure with 4 switches
(switch on 2; switch off 2).

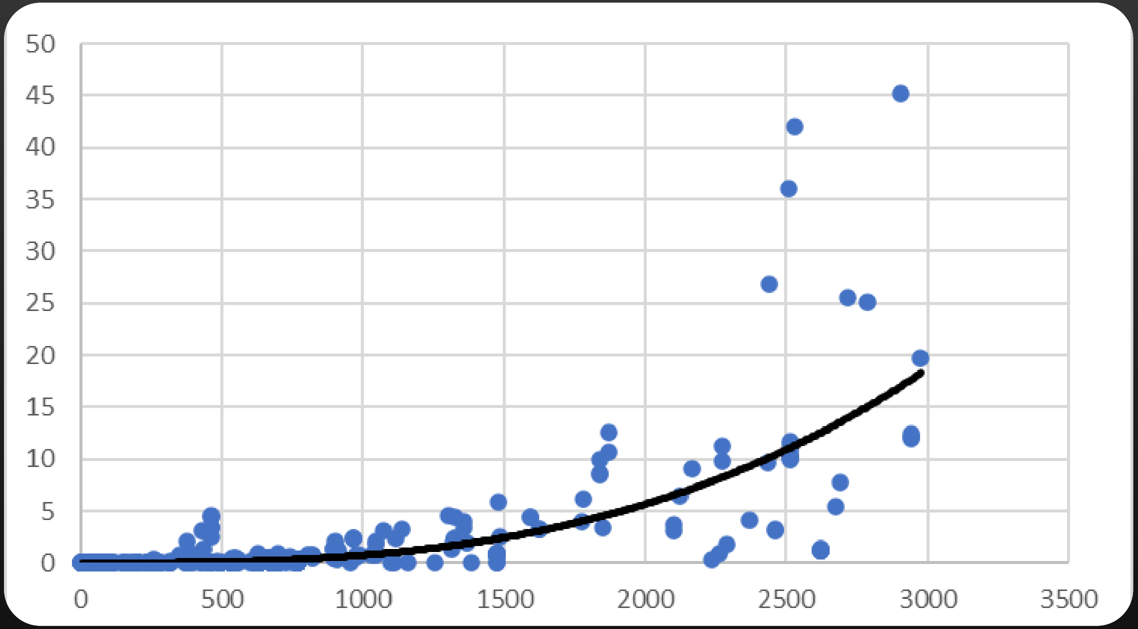




N-1 Challenge

How hard is it?

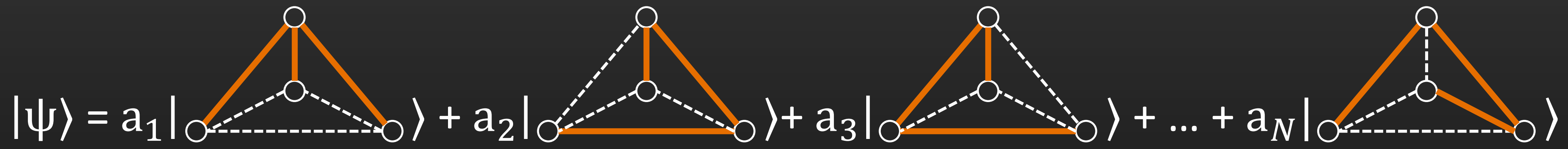
Hours



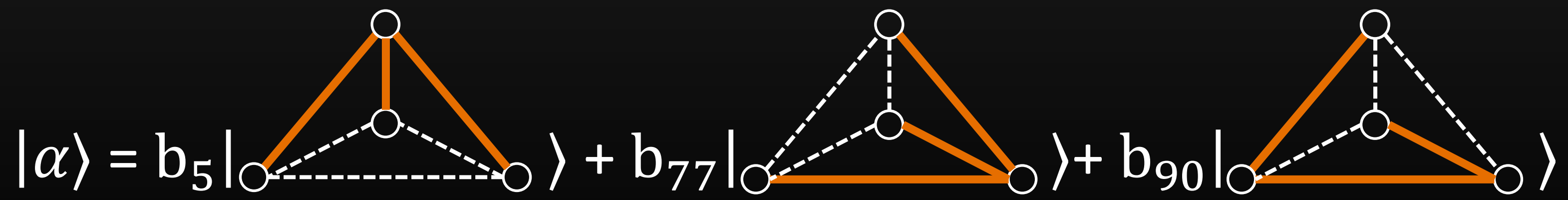
Number of network nodes

Quantum algorithm for N-1

1. Apply all possible switches using quantum parallelism.



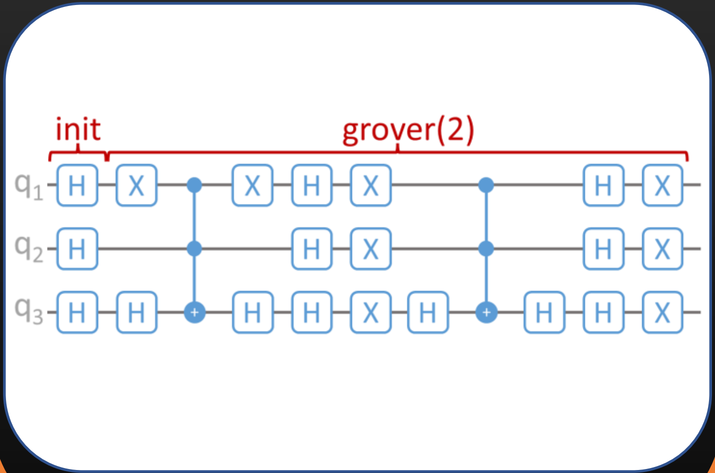
2. Apply a quantum operator to make the invalid reconfigurations vanish.



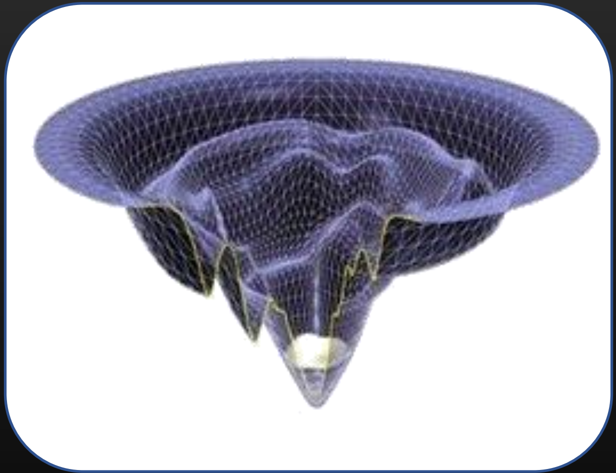
- The network is re-connected, with no cycles.
- At most 4 switches are applied.
- Load-flow constraints are met.

Which Quantum Computer

Gated Quantum Computers

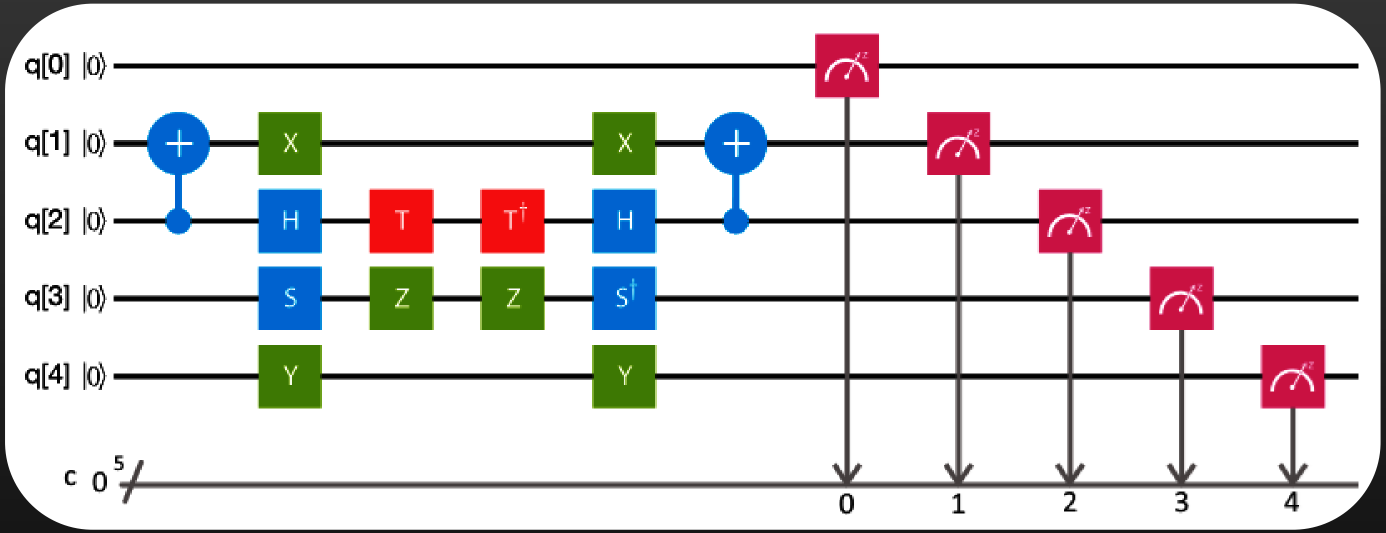


Quantum Annealers



Gate-based quantum computing

- Analogy between gate-based quantum computers and digital computers
- Gate-based quantum computers can perform universal computations



Near-term versus long-term hardware

Near-term

- Noisy-Intermediate Scale Quantum (NISQ)
- Losses are significant
  - Low decoherence time → small circuit depth
  - Gate errors + noisy measurements
  - Limited qubit connectivity
- Special-purpose devices

Long-term

- Fault Tolerant (FT)
- Logical qubits
  - Error-correcting codes are imposed on groups of qubits
  - ETA >15 years
- Universal computations

Gate-based PoC

Part 1

1. Algorithm for enumerating reconfigurations for $k = 2$
    1. Generate spanning trees which are k toggles away
2. Loop over every active edge
    1. Loop over reconfigurations found in the previous step (that deactivate the active edge being considered in this iteration)
        1. Perform a load-flow check
        2. If it passes, continue with the next active edges

Part 2

1. Repeat with $k > 2$

Gate-based PoC

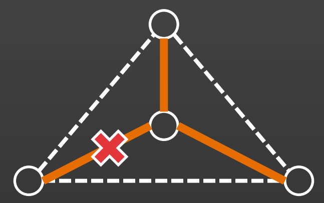
Zoom into part 2 ($k > 2$)

1. Loop over every active edge
   1. Skip if a reconfiguration for the considered active edge has already been found in a previous iteration
   2. Loop over reconfigurations found in the previous step (that deactivate the active edge being considered in this iteration)
      1. Perform a load-flow check
      2. If it passes, continue with the next active edge

QUANTUM

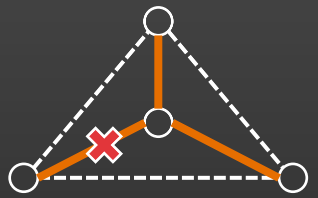
Gate-based PoC

- Assume an active edge fails

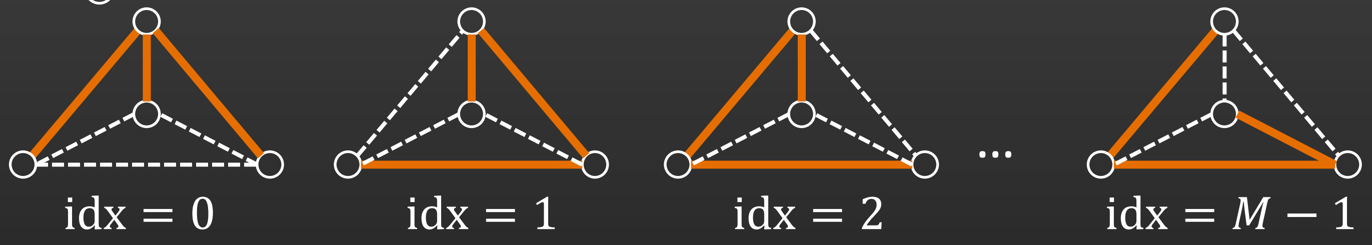


Gate-based PoC

- Assume an active edge fails

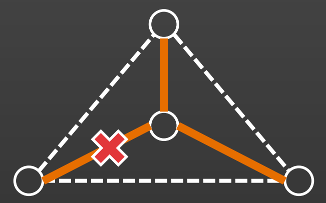


- Generate potential reconfigurations

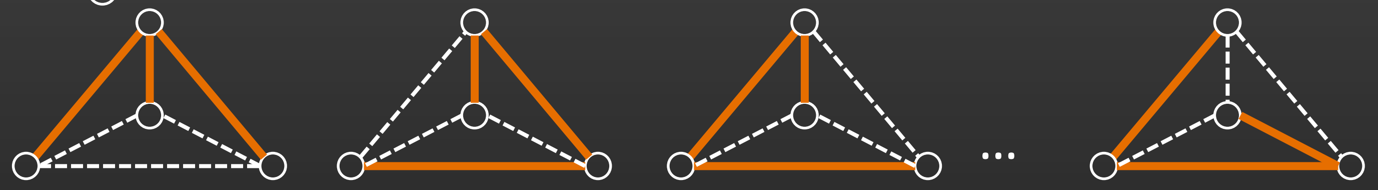


Gate-based PoC

- Assume an active edge fails

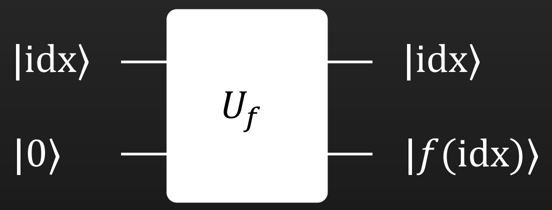


- Generate potential reconfigurations



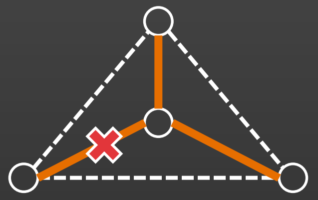
$\mathrm{idx} = 0$ $\quad \mathrm{idx} = 1$ $\quad \mathrm{idx} = 2$ $\quad \cdots \quad$ $\mathrm{idx} = M - 1$

- Define an operator U_f to check for load-flow constraints for a potential reconfiguration
  - $U_f|\mathrm{idx}\rangle|0\rangle = |\mathrm{idx}\rangle|f(\mathrm{idx})\rangle$
  - $f(\mathrm{idx}) = \begin{cases} 1 & \text{if load-flow check passes} \\ 0 & \text{otherwise} \end{cases}$

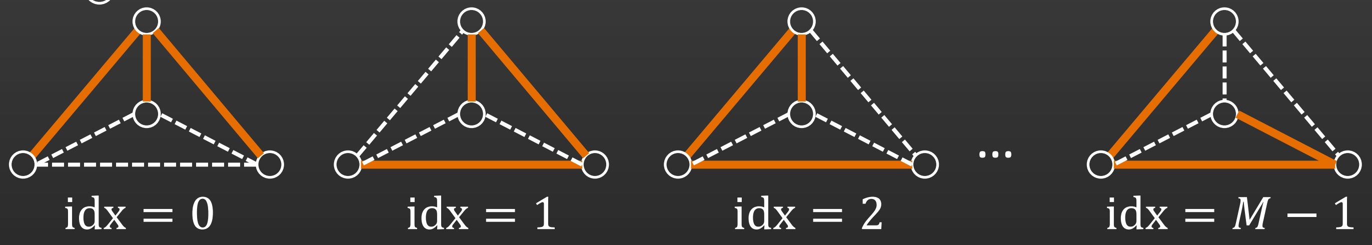


Gate-based PoC

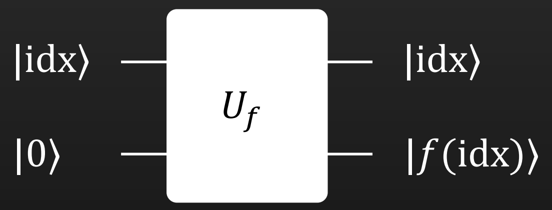
- Assume an active edge fails



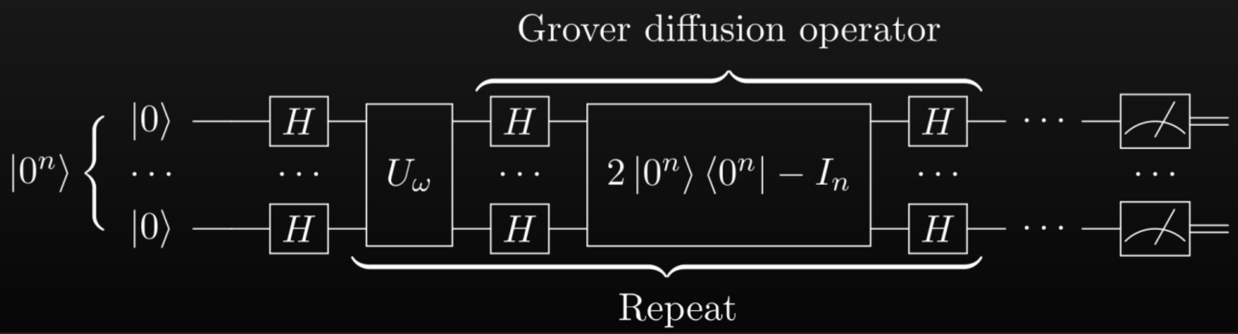
- Generate potential reconfigurations



- Define an operator U_f to check for load-flow constraints for a potential reconfiguration
  - $U_f|\mathrm{idx}\rangle|0\rangle = |\mathrm{idx}\rangle|f(\mathrm{idx})\rangle$
  - $f(\mathrm{idx}) = \begin{cases} 1 & \text{if load-flow check passes} \\ 0 & \text{otherwise} \end{cases}$

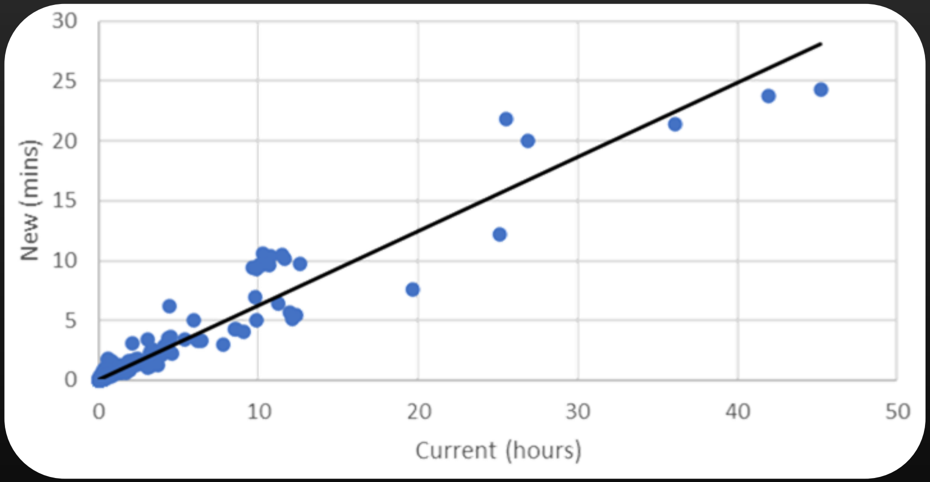


- Use Grover's algorithm to find "good" switches. Reduces complexity: $\mathcal{O}(M) \rightarrow \mathcal{O}(\sqrt{M})$.

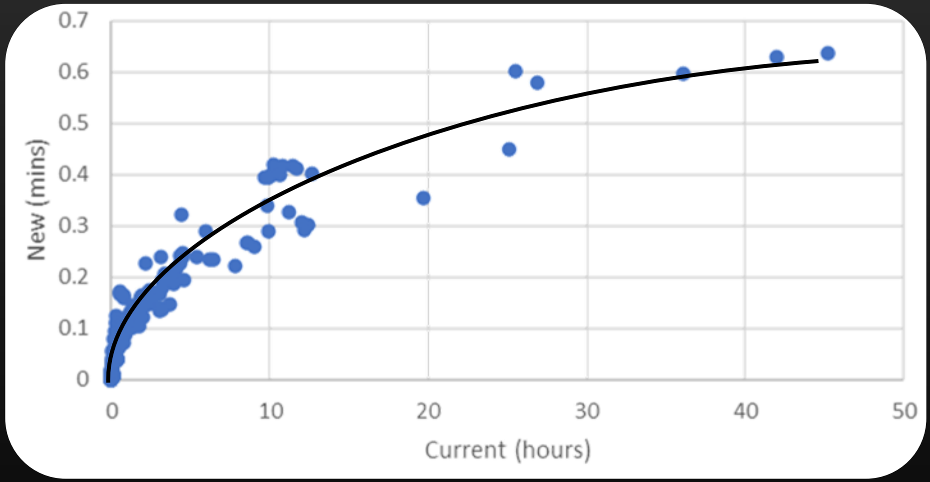


Quantum hope

Best classical speedup: linear

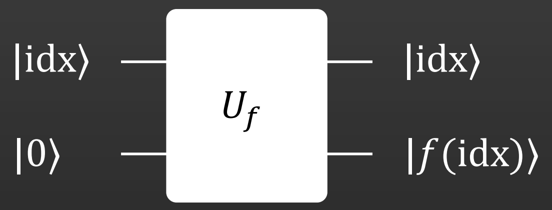


Hope for quantum speedup: quadratic

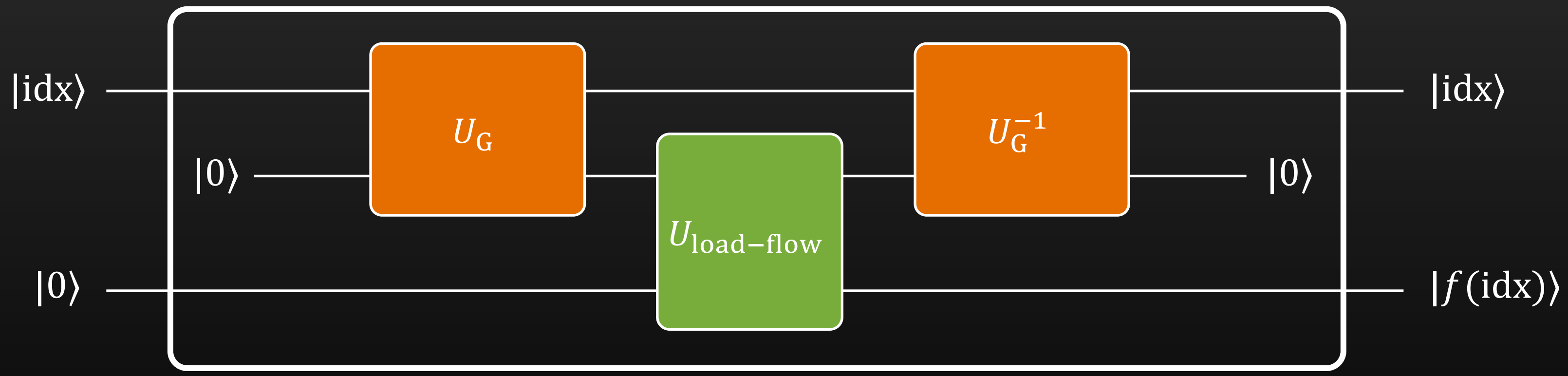


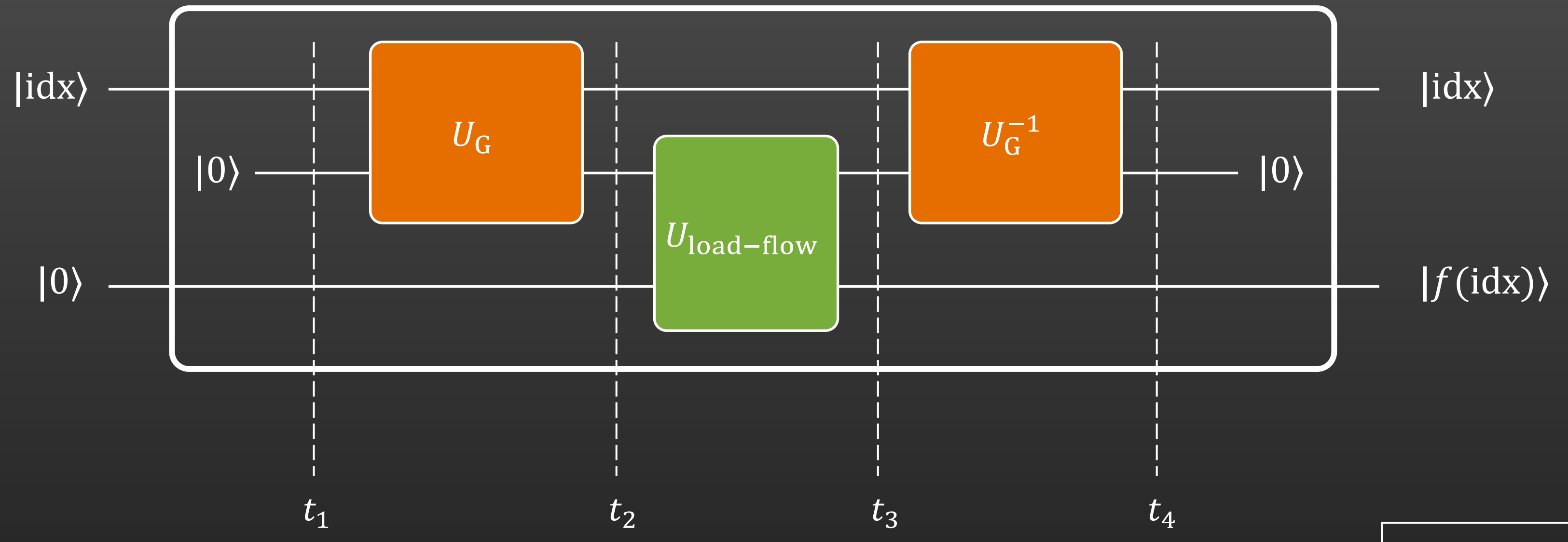
Gate-based PoC

High-level design



Detailed design





t_1: $|\text{idx}\rangle \otimes |0\rangle \otimes |0\rangle$

t_2: $|\text{idx}\rangle \otimes \left(\sum_{e \in G} |e\rangle |g(\text{idx}, e)\rangle \right) \otimes |0\rangle$

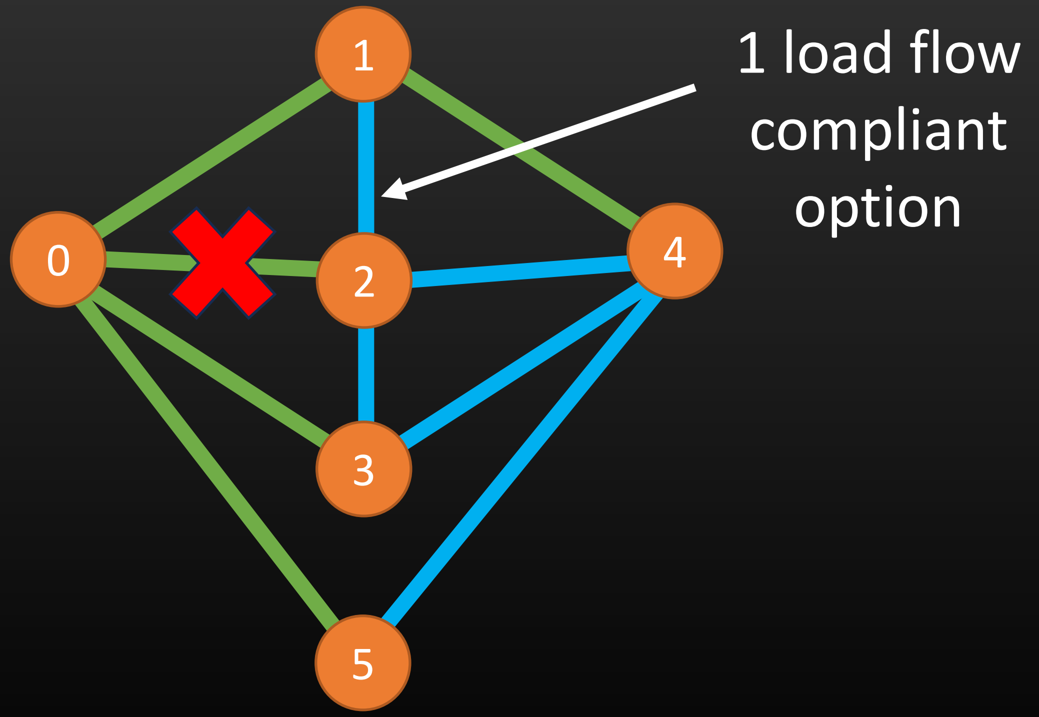
t_3: $|\text{idx}\rangle \otimes \left(\sum_{e \in G} |e\rangle |g(\text{idx}, e)\rangle \right) \otimes |f(\text{idx})\rangle$

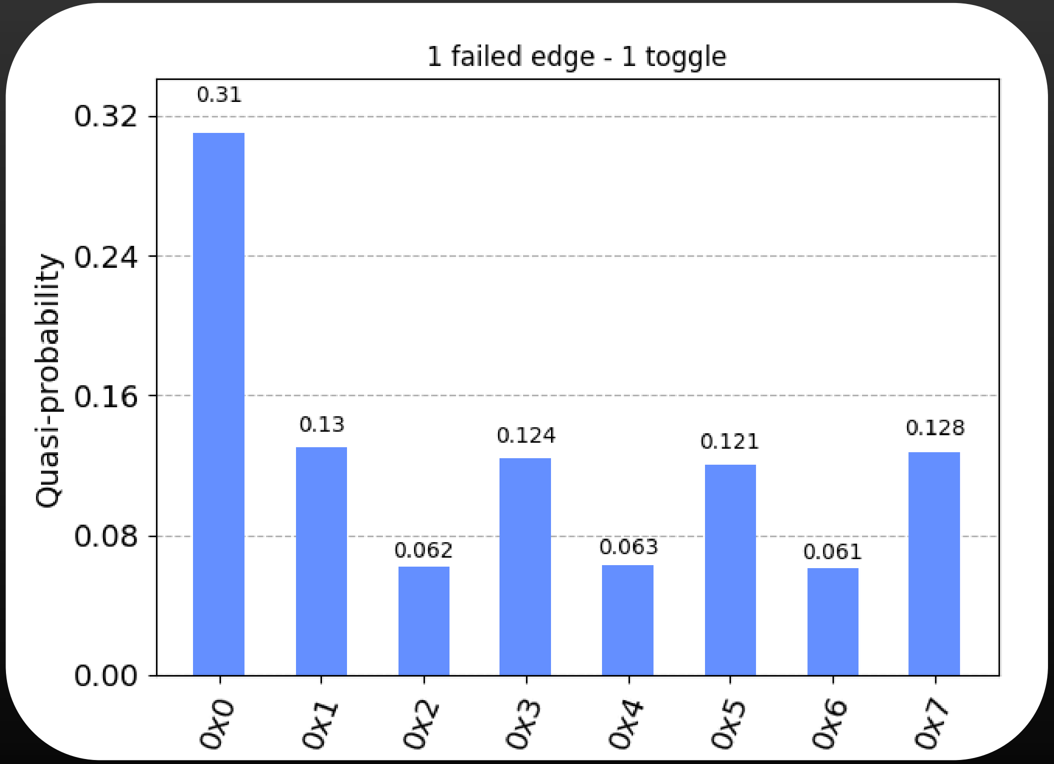
t_4: $|\text{idx}\rangle \otimes |0\rangle \otimes |f(\text{idx})\rangle$

- idx: index of spanning tree to check
- G: set of active and inactive edges
- $g(\text{idx}, e)$:
  - 1: if edge e is active in reconfiguration idx
  - 0: otherwise
- $f(\text{idx})$:
  - 1: if load-flow check passes
  - 0: otherwise

Results | k=2: 1 failure, 1 switch on

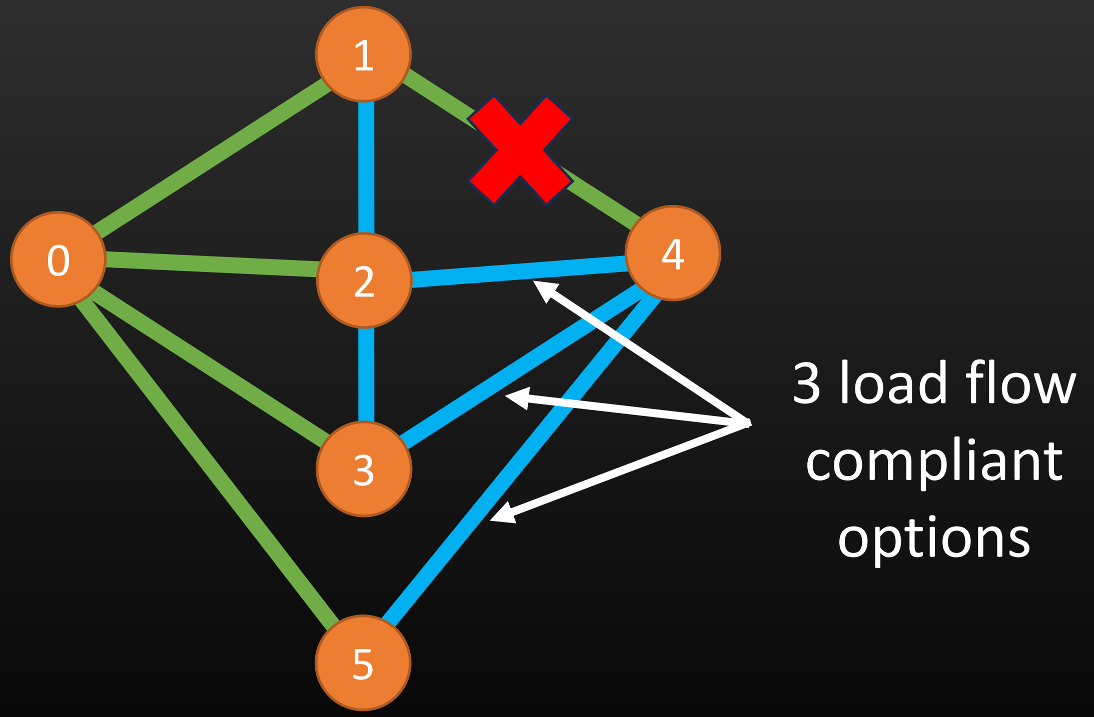
- Active edges
- Inactive toggle-edge = {0-(1,2), 1-(2,4), 2-(2,3), 3-(3,4), 4-(4,5)}

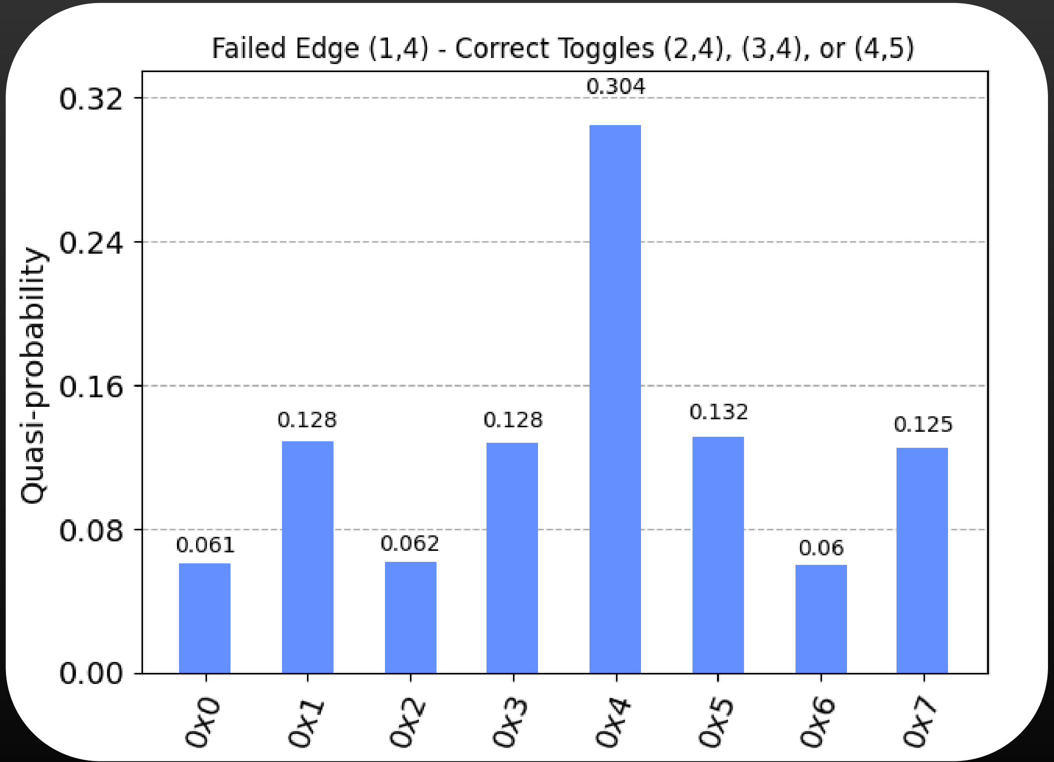




Results | k=2: 1 failure, 1 switch on

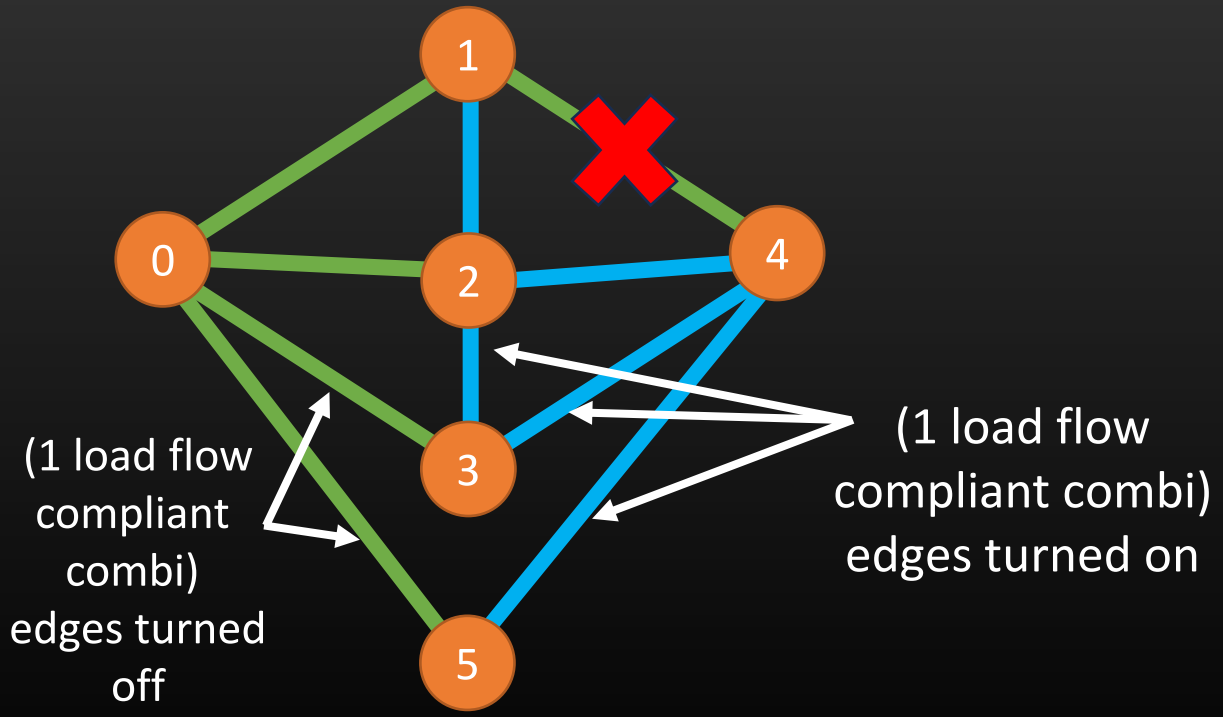
- Active edges
- Inactive toggle-edge = {0-(1,2), 1-(2,4), 2-(2,3), 3-(3,4), 4-(4,5)}

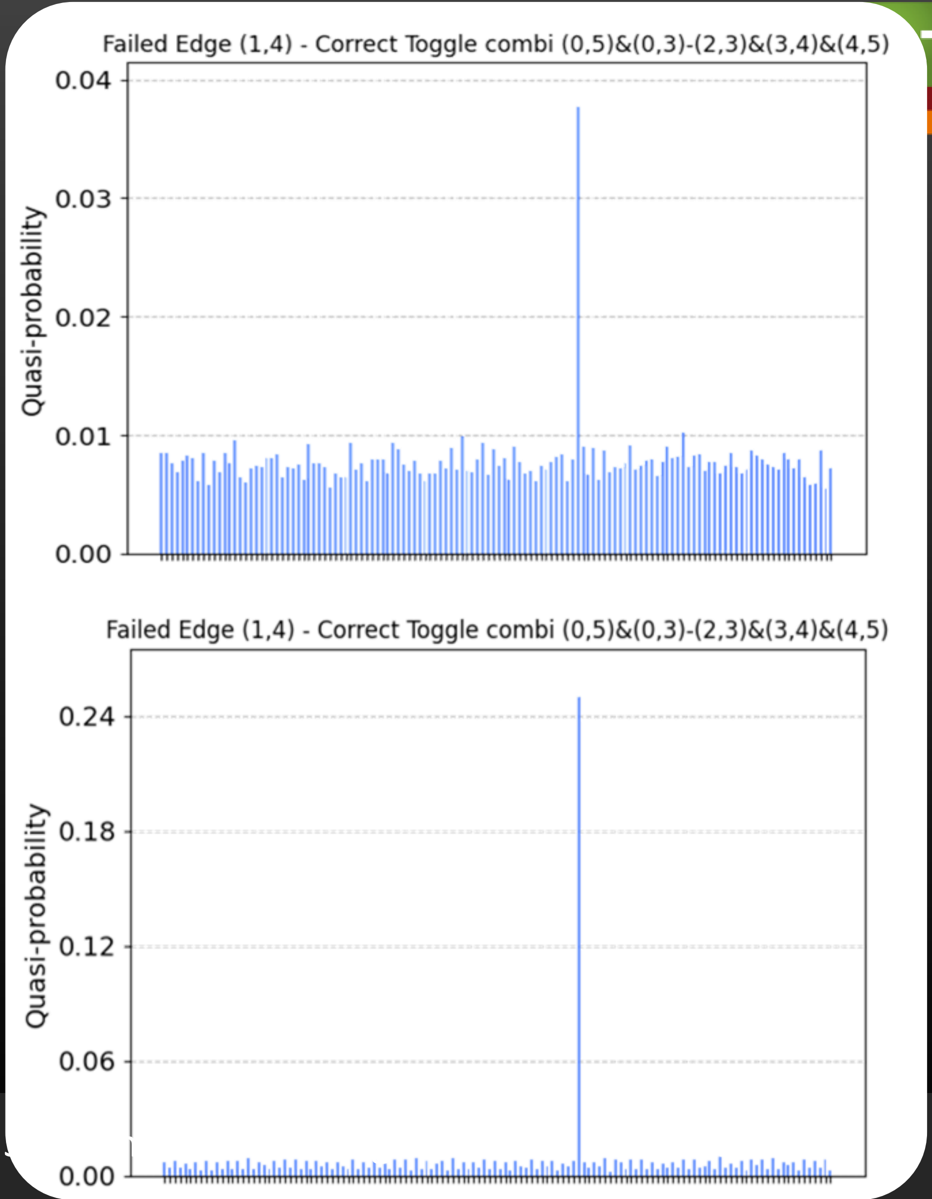




Results | k=6

- 1 failure, 2 switch offs, 3 switch ons
- Active edges, inactive edges



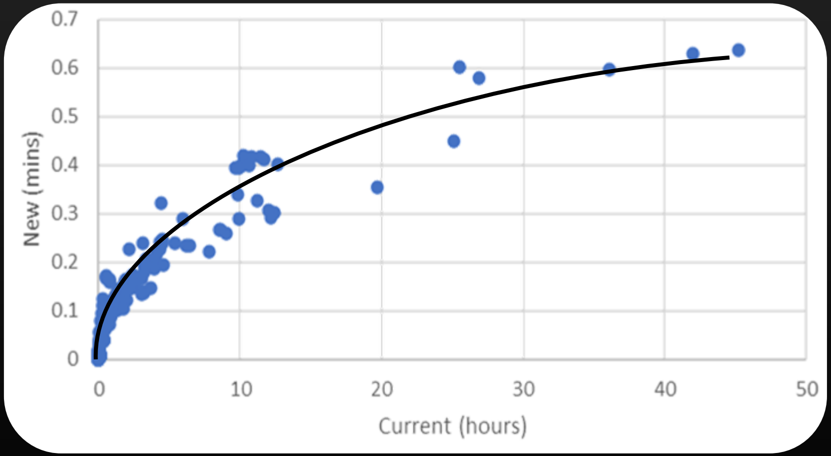


1 Grover iteration

6 Grover iterations

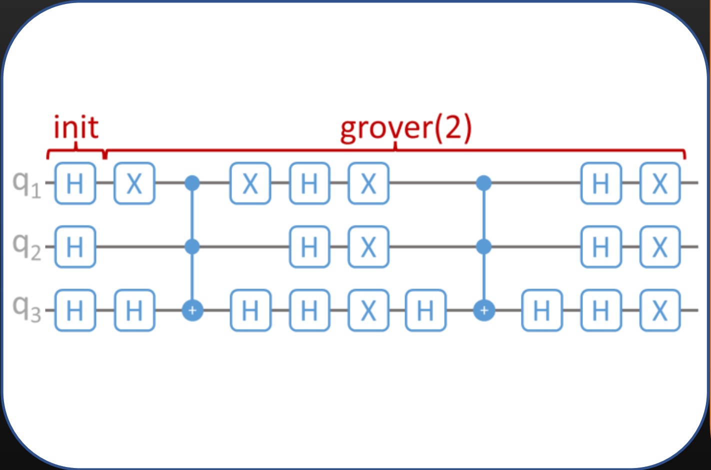
Conclusions

- Gate-based quantum approach can solve the N-1 problem
- Quadratic scaling in number of load-flow checks
- Implementation details matter for performance in practice
  - Size of search space ↔ number of Grover iterations
  - Encoding of network in quantum state
  - Load-flow check now implemented as oracle

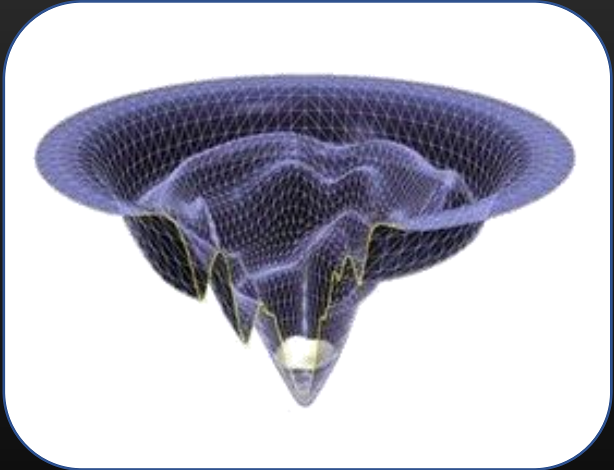


Which Quantum Computer

Gated Quantum Computers



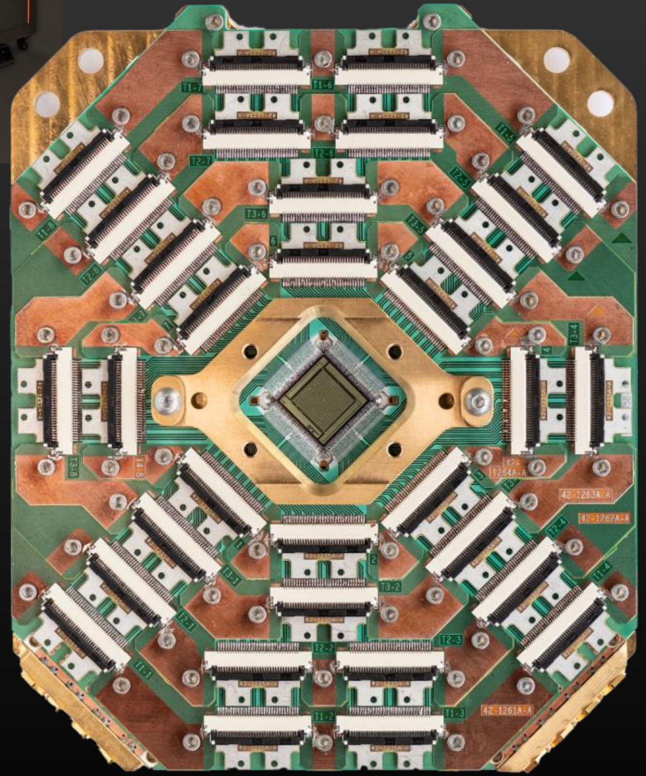
Quantum Annealers



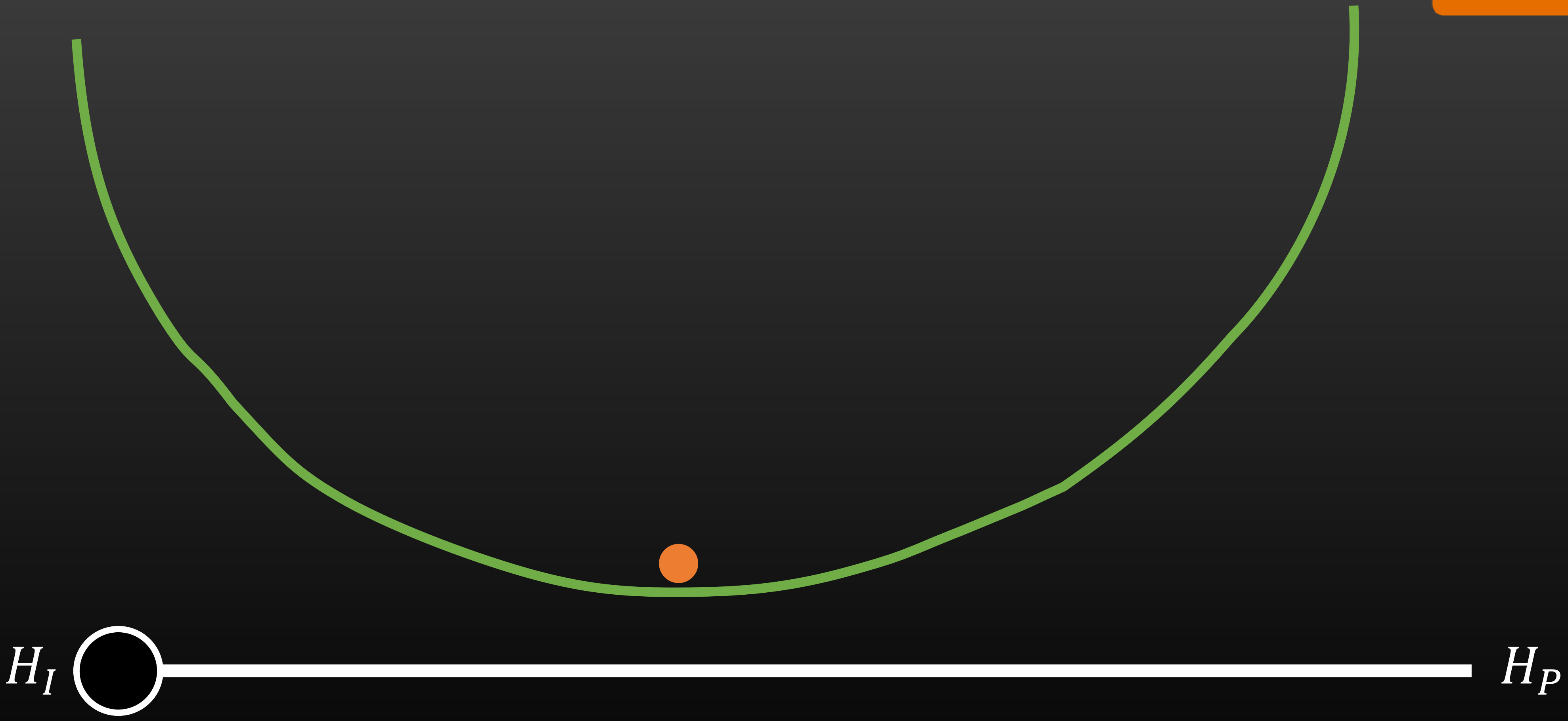
Why Quantum Annealing

- What does a quantum annealer do?
  - Solves Ising model problems
  - Solves QUBOs (Quadratic Unconstrained Binary Optimization)

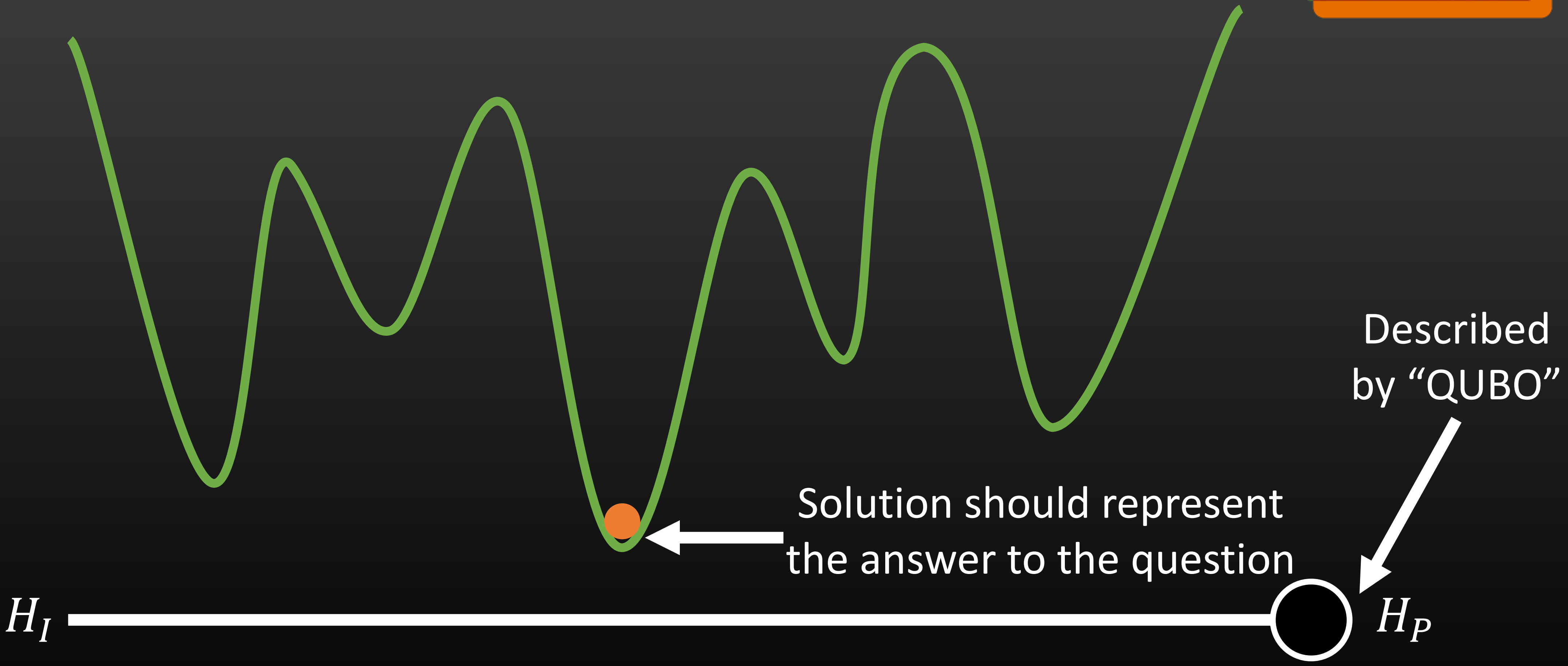
- Why do we care?
  - QUBOs are NP-Hard
  - Formulate other NP-Hard problems as QUBOs



Adiabatic Quantum Computing

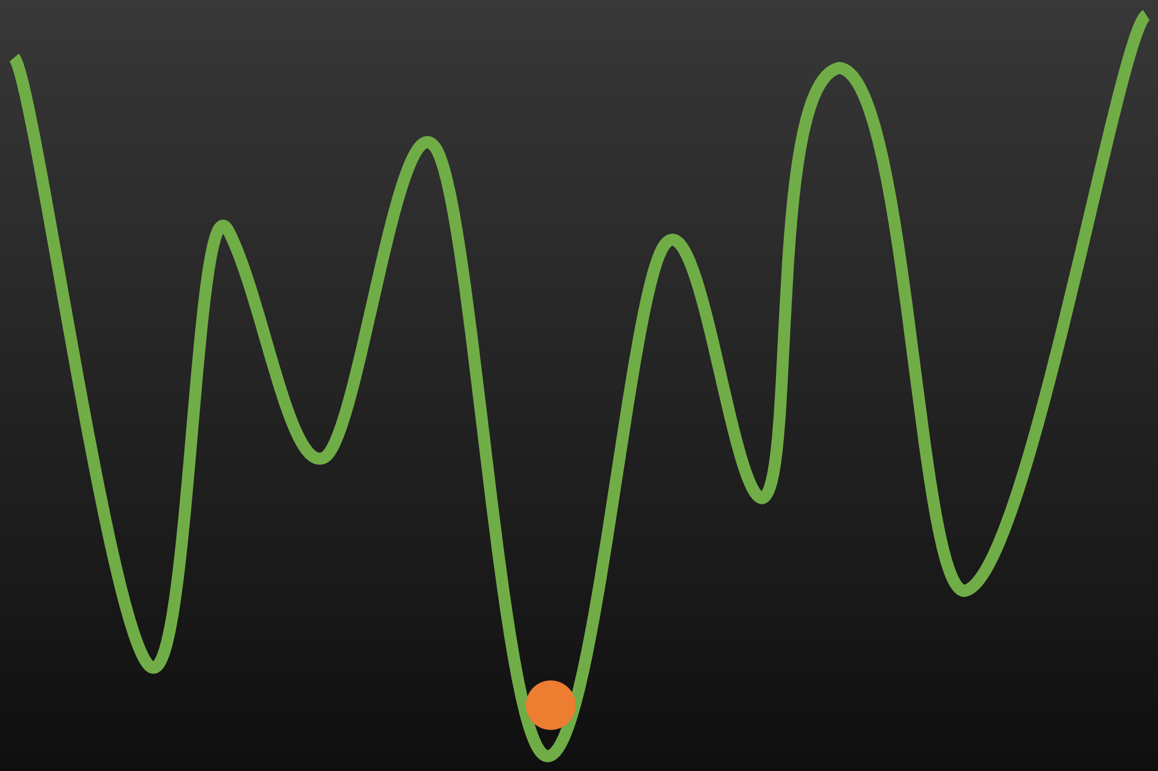


Adiabatic Quantum Computing



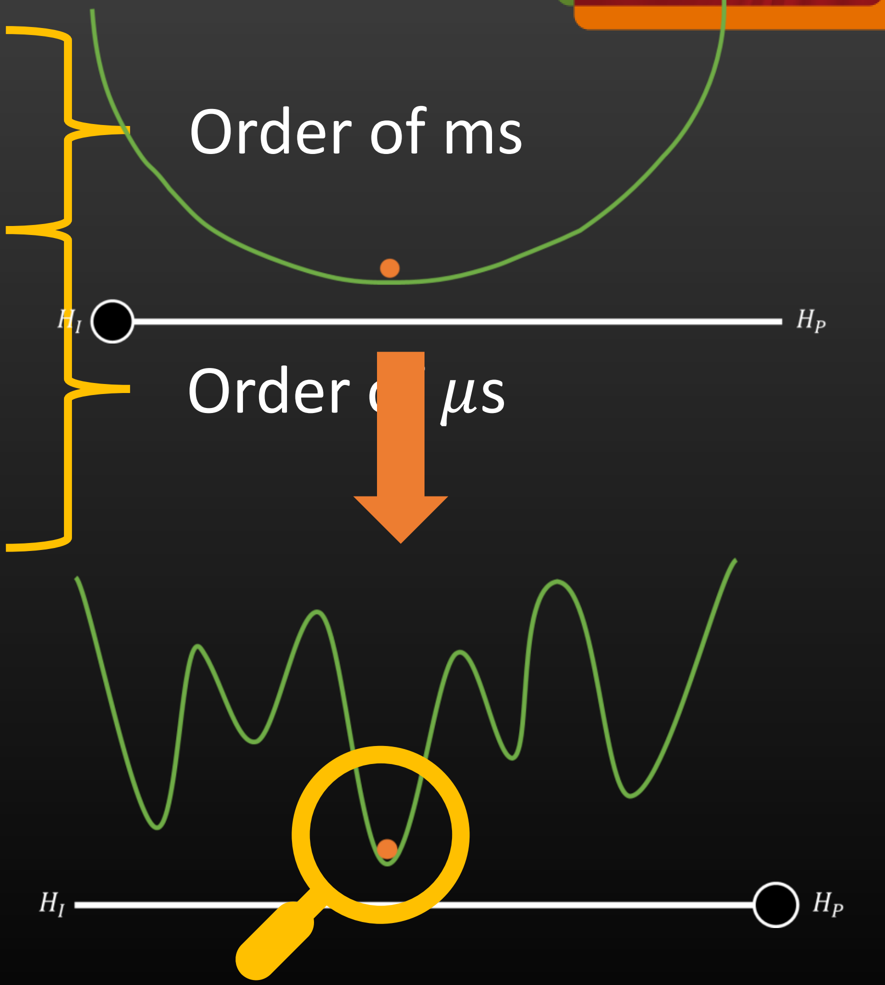
Quantum Annealing

- Same concept
  - Faster annealing schedule
  - Some noise is allowed (e.g. temperature)
- Consequence
  - (Temporarily) leave the ground state
- Stay near optimum with quantum tunnelling



Workflow (on the Quantum Annealer)

1. **Initialisation**
   - Setup of the control system
2. **Anneal**
   - $H_I \rightarrow H_P$
3. **Readout**
   - Measure the qubits
4. **Resampling**
   - Any quantum computation is probabilistic
   - Nonzero (often significant) chance to not be in the ground state



QUBO formulation

- Quadratic Unconstrained Binary Optimization

$$\min_{\boldsymbol{x}} \boldsymbol{x}^T Q \boldsymbol{x}$$

Binary vector

Real valued matrix

Quantum Annealing Based PoC

1. Search for edges with k=2 classically
2. For the remaining edges sample a QUBO which
   - Minimizes k
   - Penalizes non spanning tree configurations
   - Penalizes non load flow compliant configurations
   - Link P_{tree} to P_{load}

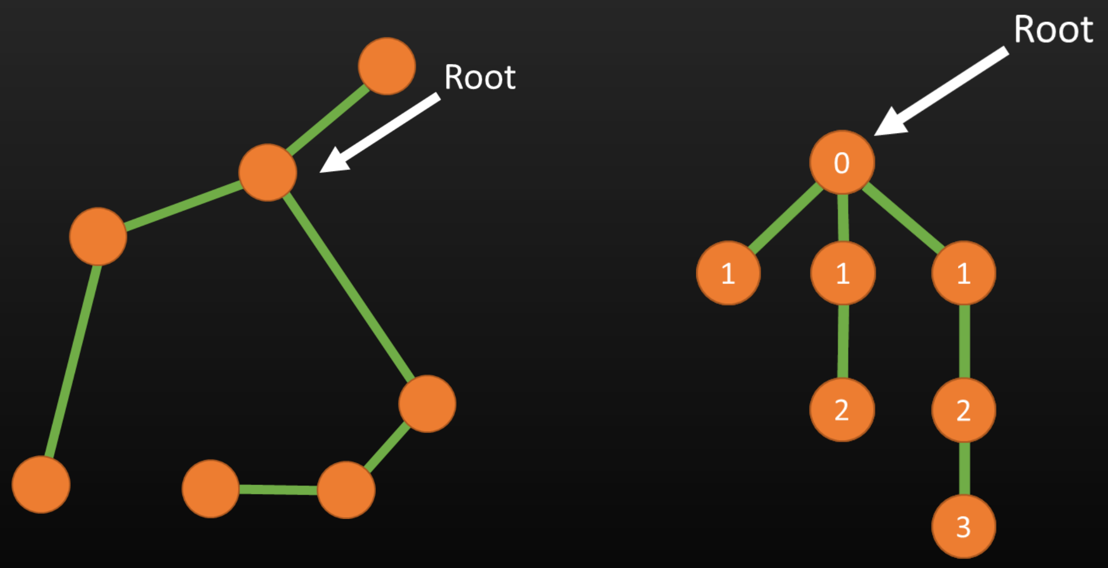
$$\min_{x \in \{0,1\}^n} H(\boldsymbol{x}) + P_{tree}(\boldsymbol{x}) + P_{load}(\boldsymbol{x}) + P_{aux}(\boldsymbol{x})$$

$$\min_{x \in \{0,1\}^n} H(\boldsymbol{x}) + P_{tree}(\boldsymbol{x}) + P_{load}(\boldsymbol{x}) + P_{aux}(\boldsymbol{x})$$

P_{tree} - Search for spanning trees

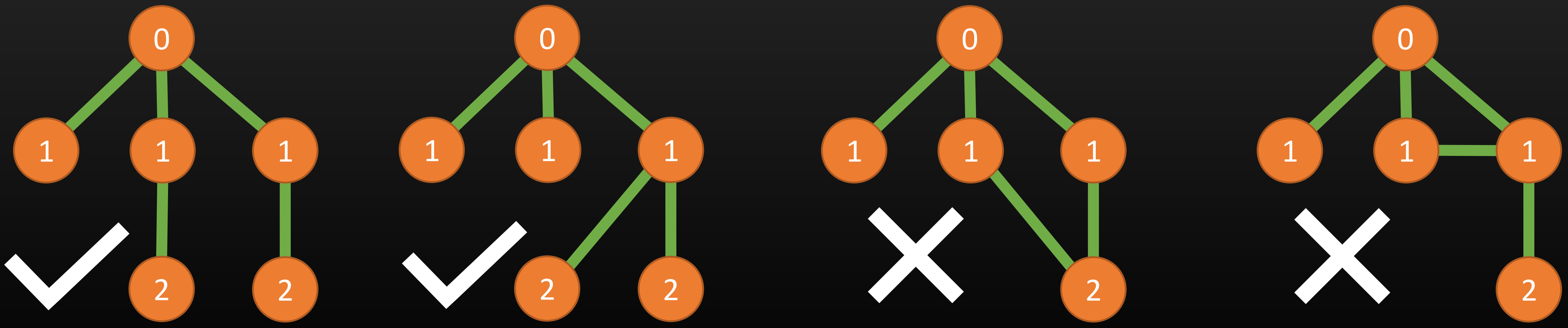
High Level Idea

- Every tree is a rooted tree
- Properties of rooted trees fit QUBO formulation



P_{tree} - Search for spanning trees

- Every node has exactly one depth
- There is exactly one root node
- Every non-root node is connected to exactly one node with lower depth
- There are no connections between nodes with the same depth



$$\min_{x \in \{0,1\}^n} H(\boldsymbol{x}) + P_{tree}(\boldsymbol{x}) + P_{load}(\boldsymbol{x}) + P_{aux}(\boldsymbol{x})$$

P_{load} - check load-flow compliance

How do classical algorithms work?

- Solve a linear system $A\boldsymbol{u} = \boldsymbol{f}$
- Check if u violated constraints

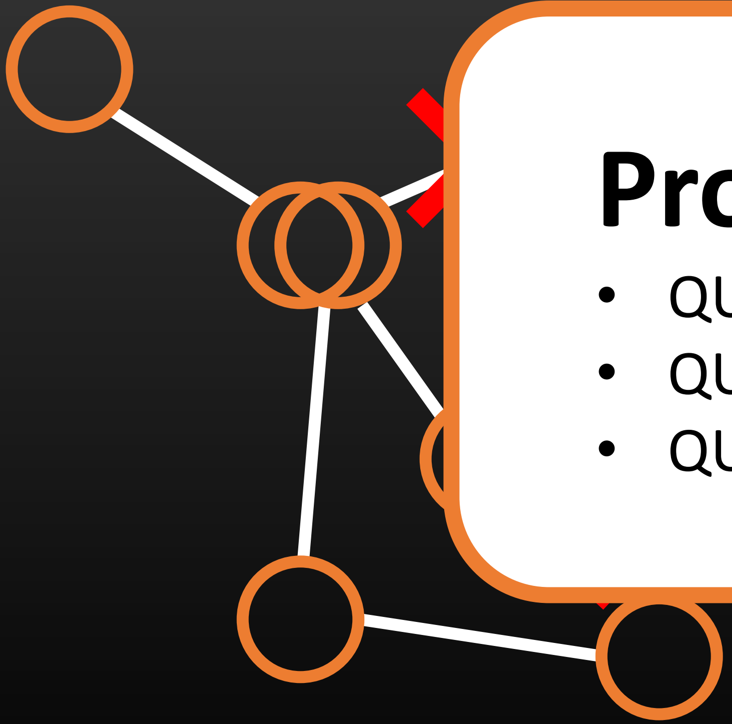
Optimization Formulation

- Encode constraints into $\hat{\mathrm{u}}(\boldsymbol{x})$
- Check if $\min_{\boldsymbol{x}} ||A\hat{\mathrm{u}}(\boldsymbol{x}) - \mathbf{f}||^2$ is close to zero
- If close to zero → load-flow compliant

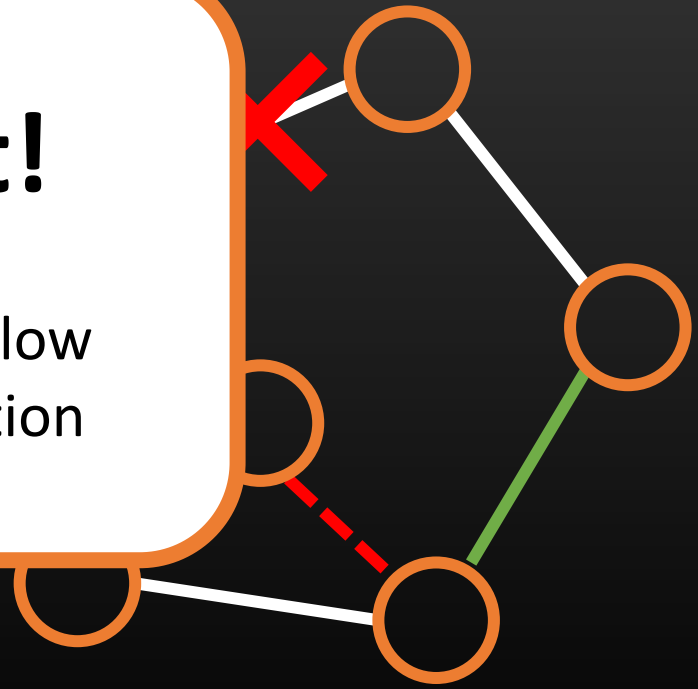
$$\min_{x \in \{0,1\}^n} H(\boldsymbol{x}) + P_{tree}(\boldsymbol{x}) + P_{load}(\boldsymbol{x}) + P_{aux}(\boldsymbol{x})$$

Results

Input Graph



Simulated Annealing Output



Proof of Concept!

- QUBO finds spanning trees
- QUBO also checks the load-flow
- QUBO gives the reconfiguration

Why just Simulated Annealing?

$$\min_{x\in\{0,1\}^n} H(\boldsymbol{x}) + P_{tree}(\boldsymbol{x}) + P_{load}(\boldsymbol{x}) + P_{aux}(\boldsymbol{x})$$

$$\lambda_1\left(\sum_{v\in V}\sum_{i=0}^{I-3} x_{v,i}(1 - x_{v,i+1}\right.$$

$$+\lambda_2\left(1-\sum_{v\in V} x_{v,0}\right)^2$$

$$+\lambda_3\sum_{v\in V}\sum_{i=1}^{I-1}\left(x_{v,i} - x_{v,}\right.$$

$$+\lambda_4\sum_{(v,u)\in E}\sum_{i=0}^{I-2}(y_{vu,i} - y_{vu,}$$

$$x_{v,i} + x_{v,i-1} - x_{u,i+1} + x_{u,i})$$

Current QA hardware is very sensitive to these hyperparameters (compared to classical solvers)

Results

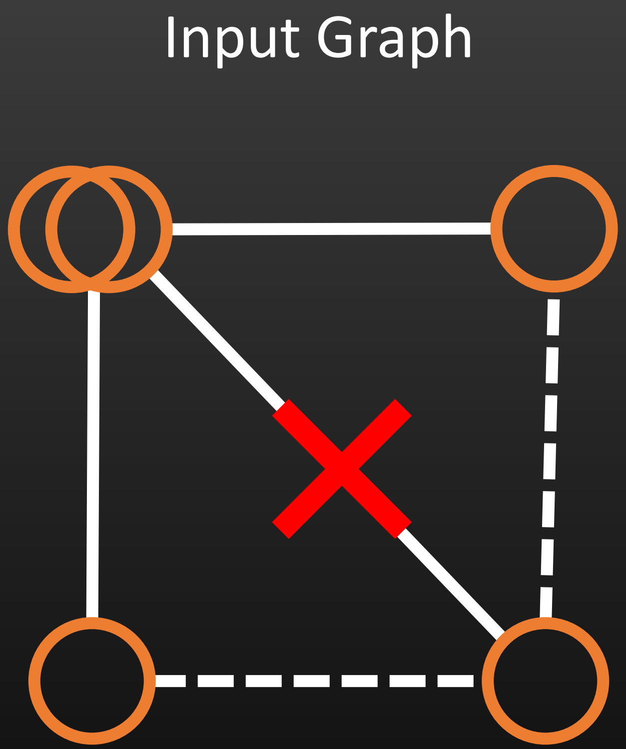
Quantum Annealing

- Downsize the problem
  - 4 Nodes
- No Load-flow check

$$
\begin{aligned}
&\lambda_1\left(\sum_{v\in V}\sum_{i=0}^{I-3} x_{v,i}\left(1-x_{v,i+1}\right)+\sum_{(v,u)\in E}\sum_{i=0}^{2I-3} y_{vu,i}\left(1-y_{vu,i+1}\right)\right)\\
&+\lambda_2\left(1-\sum_{v\in V} x_{v,0}\right)^2\\
&+\lambda_3\sum_{v\in V}\sum_{i=1}^{I-1}\left(x_{v,i}-x_{v,i-1}-\sum_{u:(v,u)\in E} y_{vu,i-1}-y_{vu,i-2}-\sum_{u:(u,v)\in E} y_{uv,I+i-2}-y_{uv,I+i-3}\right)^2\\
&+\lambda_4\sum_{(v,u)\in E}\sum_{i=0}^{I-2}\left(y_{vu,i}-y_{vu,i-1}\right)\left(2-x_{v,i+1}+x_{v,i}-x_{u,i}+x_{u,i-1}\right)+\left(y_{vu,I+i-1}-y_{vu,I+i-2}\right)\left(2-x_{v,i}+x_{v,i-1}-x_{u,i+1}+x_{u,i}\right)
\end{aligned}
$$

$$
\min_{x\in\{0,1\}^n} H(\boldsymbol{x}) + P_{tree}(\boldsymbol{x}) \cancel{+ P_{load}(\boldsymbol{x}) + P_{aux}(\boldsymbol{x})}
$$

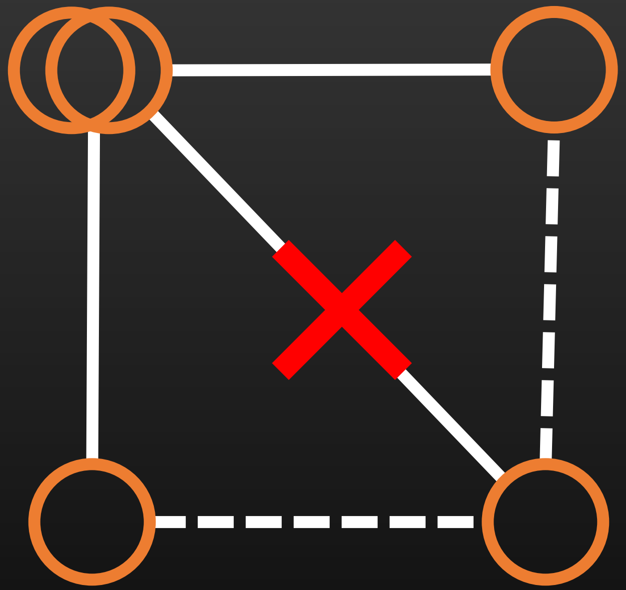
Quantum Annealing



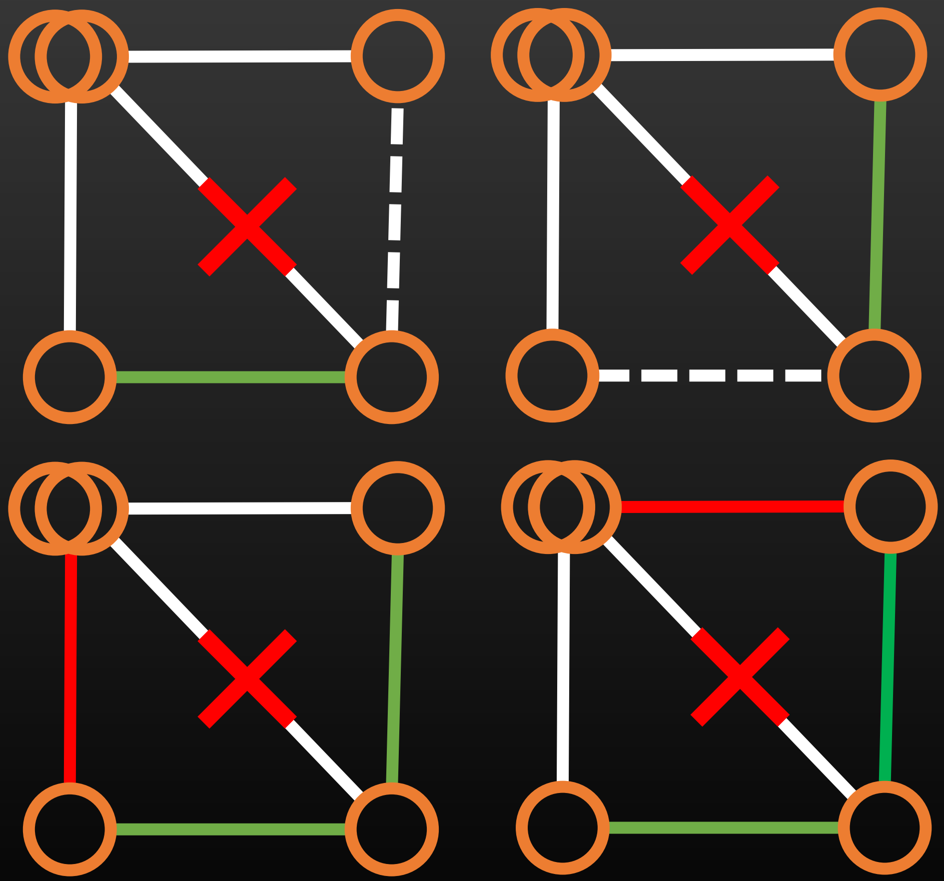
- Needs 46 qubits
- $2^{46} \approx 7 \cdot 10^{13}$ (70 trillion) possible outcomes
- $P_{tree}(\boldsymbol{x}) = 0$ for **4** outcomes

Quantum Annealing

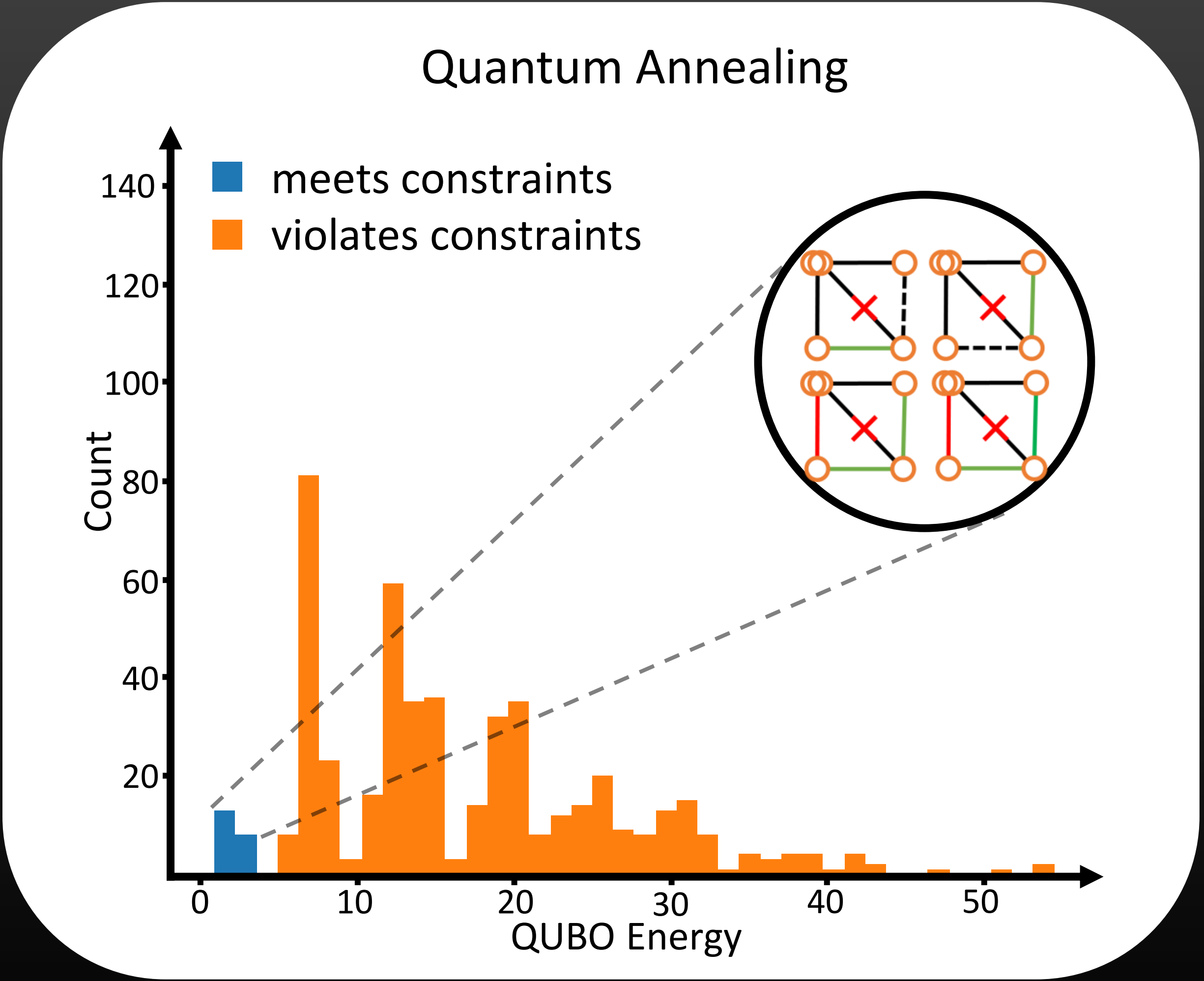
Input Graph



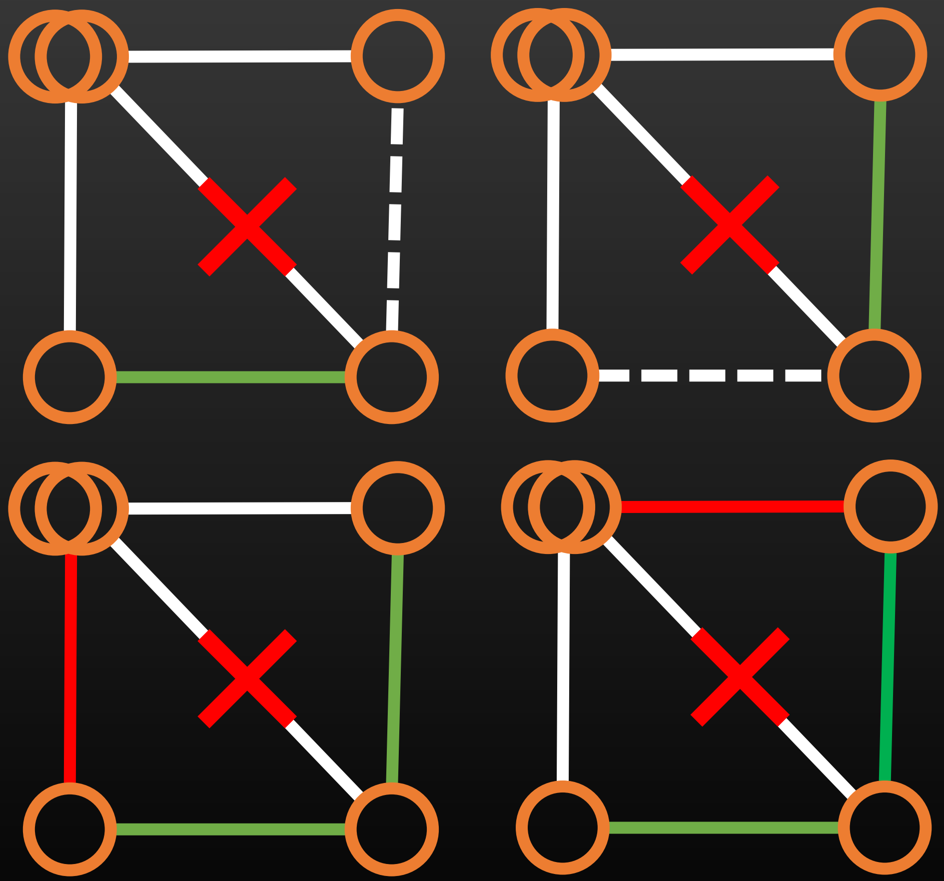
Quantum Annealing Output



How often did we find them?



Quantum Annealing Output



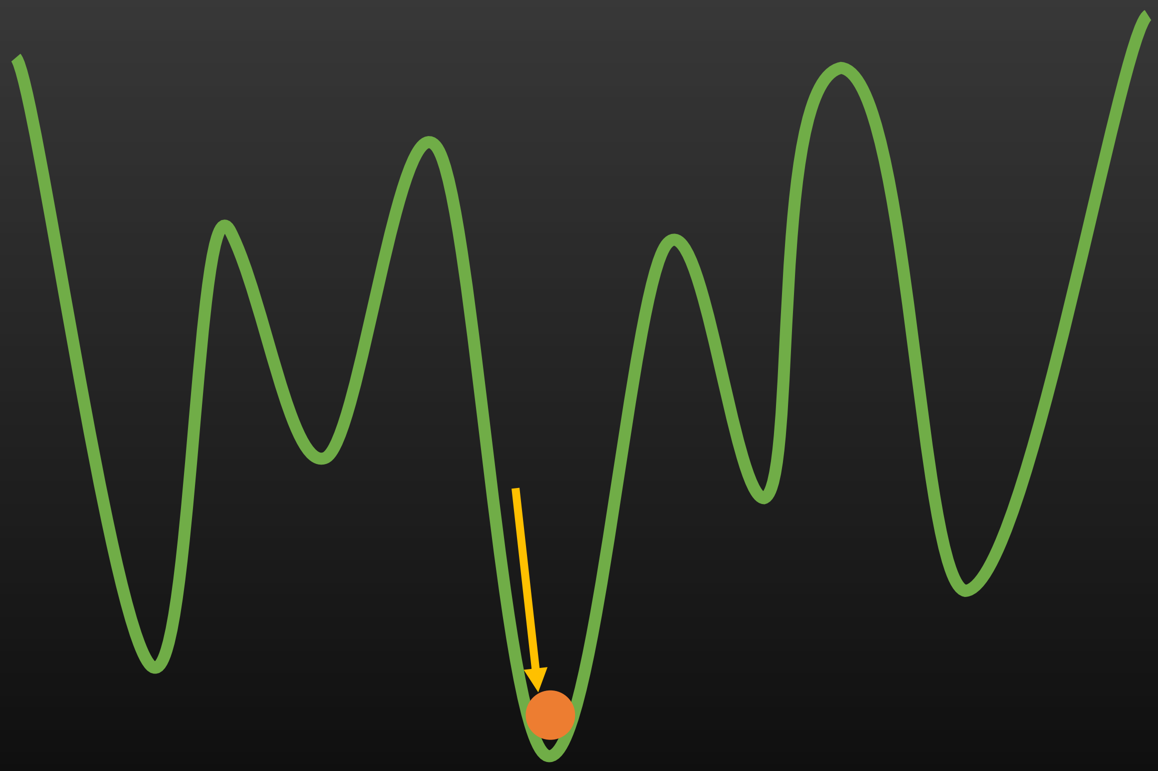
Can we improve this?

- Yes!

Common misconception:

- Quantum Annealing always returns a (local) minimum

- Greedy postprocessing
  - Steepest Descent



Conclusion and Outlook

- Formulated a "N-1 QUBO"
- Successfully solved the QUBO with Simulated Annealing
- Solved part of the QUBO with Quantum Annealing

Quantum Annealing for N-1 is promising

- Challenges:
  - Problem size
  - Choice of hyper-parameters

Current hardware limitations

Outlook: Challenges and Hardware

- Problem size
  - Number of qubits
  - Number of couplers
- Choice of hyper-parameters
  - Quality of the Qubits ?

Questions?